Λειτουργικά Συστήματα Υπολογιστών
7ο Εξάμηνο, ΣΗΜΜΥ ΕΜΠ
17/11/2016

Αναφορά 2ης Άσκησης

Ομάδα E15
Γαβαλάς Νικόλαος 03113121
Καραΐσκου Κωνσταντίνα 03113127

Άσκηση 1.1

Source Code:

```c
#include <unistd.h>
#include <stdlib.h>
#include <stdio.h>
#include <signal.h>
#include <assert.h>
#include <sys/wait.h>

#include "proc-common.h"

#define SLEEP_SEC 5
#define SLEEP_LESS 3


int main(void){

        pid_t p;
        int status;

        fprintf(stderr, "TreeRoot, PID = %ld: Creating A...\n",(long)getpid());

        p = fork();                     // δημιουργία διεργασίας
        if (p < 0) {
                /* fork failed */
                perror("fork");
```

```
                exit(1);
        }
        if (p == 0) {
                /* In child process A*/
                change_pname("A");              //  ονόμασε την A

                fprintf(stderr, "A, PID = %ld: Creating B...\n",(long)getpid());

                p = fork();         //  δημιουργία της διεργασίας B, παιδί της A
                if (p < 0) {
                        /* fork failed */
                        perror("fork");
                        exit(1);
                }
                if (p == 0) {
                        /* In child process B*/
                        change_pname("B");          // ονόμασε τη B

                        fprintf(stderr, "B, PID = %ld: Creating D...\n",
                        (long)getpid());

                        p = fork(); //  δημιουργία της διεργασίας D, παιδί της B
                        if (p < 0) {
                                /* fork failed */
                                perror("fork");
                                exit(1);
                        }
                        if (p == 0) {
                                /* In child process D */
                                change_pname("D"); //  ονόμασε τη D

                                printf("D: Sleeping...\n");
                                sleep(SLEEP_SEC); //  η D κοιμάται για 5 δευτερόλεπτα
(είναι  φύλλο)

                                printf("D: Waking up..\n");
                                exit(13);// η D τερματίζει με κωδικό επιστροφής 13
                        }
                       /* Back in B */
```

```
                printf("B, PID = %ld: Created D with PID = %ld, waiting for
                it to terminate...\n", (long)getpid(), (long)p);

                p = wait(&status);    // η B περιμένει μέχρι να τερματιστεί το παιδί
της
                explain_wait_status(p, status);// βγάζει το κατάλληλο μήνυμα
ανάλογα με το επιστρεφόμενο status της wait

                printf("B: All done, exiting...\n");
                exit(19); // η B τερματίζει με κωδικό επιστροφής 19
        }
        /*Back in A */

        printf("A, PID = %ld: Created B with PID = %ld, waiting for it to
        terminate...\n", (long)getpid(), (long)p);

        fprintf(stderr, "A, PID = %ld: Creating C...\n", (long)getpid());

        p = fork();           // δημιουργία της διεργασίας C, παιδί της A
                if (p < 0) {
                        /* fork failed */
                        perror("fork");
                        exit(1);
                }
                if (p == 0) {
                        /* In child process C */
                        change_pname("C");          // ονόμασε τη C

                        printf("C: Sleeping...\n");
                        sleep(SLEEP_SEC);     // η C κοιμάται για 5 δευτερόλεπτα
(είναι φύλλο)

                        printf("C: Waking up..\n");
                        exit(17);             // η C τερματίζει με κωδικό
επιστροφής 17
                }
        /* Back in A */

        printf("A, PID = %ld: Created C with PID = %ld, waiting for it to
        terminate...\n", (long)getpid(), (long)p);
```

```
                //wait for both B and C
                p = wait(&status); // η A περιμένει να τερματιστούν τα δύο παιδιά της
                explain_wait_status(p, status);  // βγάζει το κατάλληλο μήνυμα ανάλογα
με το επιστρεφόμενο status της wait

                p = wait(&status);
                explain_wait_status(p, status);
                printf("Parent A: All done, exiting...\n");
                exit(16);               // η A τερματίζει με κωδικό επιστροφής 16
        }
       /* Back in TreeRoot */

        change_pname("TreeRoot");      // ονόμασε τη main διεργασία TreeRoot

        printf("TreeRoot, PID = %ld: Created A with PID = %ld, waiting for it to
        terminate...\n", (long)getpid(), (long)p);

        sleep(SLEEP_LESS); // η TreeRoot κοιμάται 3 δευτερόλεπτα για να δημιουργηθούν όλες
οι διεργασίες

        show_pstree(p);           // εμφάνισε το δέντρο διεργασιών

        p = wait(&status); // η TreeRoot περιμένει να τερματιστεί η A
        explain_wait_status(p, status);// βγάζει το κατάλληλο μήνυμα ανάλογα με το
επιστρεφόμενο status της wait

        printf("TreeRoot: All done, exiting...\n");

        return 0;
}
```

Παράδειγμα Εκτέλεσης:

```
oslabe15@anafi:~/erg2/forktree$ ./ask1
TreeRoot, PID = 3581: Creating A...
TreeRoot, PID = 3581: Created A with PID = 3582, waiting for it to terminate...
A, PID = 3582: Creating B...
```

```
A, PID = 3582: Created B with PID = 3583, waiting for it to terminate...
A, PID = 3582: Creating C...
A, PID = 3582: Created C with PID = 3584, waiting for it to terminate...
C: Sleeping...
B, PID = 3583: Creating D...
B, PID = 3583: Created D with PID = 3585, waiting for it to terminate...
D: Sleeping...


A(3582)─┬─┬─B(3583)─────D(3585)
        └─└─C(3584)



C: Waking up..
My PID = 3582: Child PID = 3584 terminated normally, exit status = 17
D: Waking up..
My PID = 3583: Child PID = 3585 terminated normally, exit status = 13
B: All done, exiting...
My PID = 3582: Child PID = 3583 terminated normally, exit status = 19
Parent A: All done, exiting...
My PID = 3581: Child PID = 3582 terminated normally, exit status = 16
TreeRoot: All done, exiting...
```

Απαντήσεις στις ερωτήσεις:

1. Τι θα γίνει αν τερματίσετε πρόωρα τη διεργασία A, δίνοντας kill -KILL <pid>, όπου <pid> το Process ID της;

Αν η Α τερματιστεί πρόωρα, τότε δεν θα προλάβει να κάνει wait() για τα παιδιά της, οπότε θα μείνουν “ορφανά” όπως λέμε. Σε τέτοια περίπτωση γίνονται παιδιά της διεργασίας init, οι οποία είναι σε αέναο loop που κάνει wait (για τέτοιες περιπτώσεις), οπότε τα παιδιά θα τερματίσουν από εκεί.

2. Τι θα γίνει αν κάνετε show_pstree(getpid()) αντί για show_pstree(pid) στη main(); Ποιες επιπλέον διεργασίες φαίνονται στο δέντρο και γιατί;

Κάνοντας το αυτό, το δέντρο διεργασιών θα τυπωθεί με ρίζα τη main και όχι την διεργασία “A”. Αυτό έχει σαν αποτέλεσμα να φαίνονται και δύο άλλες διεργασίες, συγκεκριμένα η sh και η pstree. Η sh είναι η διεργασία του φλοιού (shell, το

command line interface) και η pstree είναι διεργασία που τρέχει από τον φλοιό και είναι εκείνη που τυπώνει το δέντρο διεργασιών από δοθείσα ρίζα. Επομένως συμπεραίνουμε ότι η συνάρτηση show_pstree() χρησιμοποιεί την διεργασία pstree. Παράδειγμα:

```
Father(2909)─┬─A(2910)─┬─B(2911)──C(2913)
             │         └─D(2912)
             └─sh(2914)──pstree(2915)
```

3. Σε υπολογιστικά συστήματα πολλαπλών χρηστών, πολλές φορές ο διαχειριστής θέτει όρια στον αριθμό των διεργασιών που μπορεί να δημιουργήσει ένας χρήστης. Γιατί;

Γιατί αν ο χρήστης είχε δικαιώματα να φτιάξει πολλές διεργασίες, θα μπορούσε πολύ εύκολα να εξαντλήσει τους πόρους του συστήματος, όπως τη μνήμη, δημιουργώντας έναν μεγάλο αριθμό από αυτές και να προκαλέσει προβλήματα στους υπόλοιπους χρήστες ή και να κρασάρει το υπολογιστικό σύστημα (fork bomb).

Άσκηση 1.2

Source Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <assert.h>
#include <sys/wait.h>

#include "proc-common.h"
#include "tree.h"


void create(struct tree_node *root);


int main(int argc, char *argv[]){
        struct tree_node *root;
        // αν τα αρχεία εισόδου είναι διάφορα του 2 τότε τύπωσε το ακόλουθο μήνυμα
        if (argc != 2) {
                fprintf(stderr, "Usage: %s <input_tree_file>\n\n", argv[0]);
```

```
                exit(1);
        }

        root = get_tree_from_file(argv[1]);   // ο δείκτης root δείχνει στη ρίζα του δέντρου
που διαβάζουμε από το αρχείο εισόδου

        if (root!=NULL) {

                print_tree(root);       // τύπωσε τα στοιχεία του δέντρου
                printf("\n");

                pid_t pid;
                int status;

                fprintf(stderr, "Parent Father, PID = %ld: Creating
                child...\n",(long)getpid());

                pid = fork();       // δημιούργησε το παιδί που αντιστοιχεί στη ρίζα του
δέντρου
                if (pid < 0) {
                        perror("main: fork");
                        exit(1);
                }
                if (pid == 0) {
                       /* Child */
                        change_pname(root->name);       // ονόμασέ το με το όνομα του
κόμβου
                        create(root);
                        exit(101);
                }

                sleep(3);         // η αρχική διεργασία κοιμάται περιμένοντας να
δημιουργηθούν όλες οι διεργασίες

                show_pstree(pid);             // εμφάνισε το δέντρο

                wait(&status);          // περιμένει να τερματιστεί το παιδι της
                explain_wait_status(pid, status);
        }
        else          // αν το δέντρο είναι κενό τύπωσε το αντίστοιχο μήνυμα
                printf("The input tree file is empty!\n");
```

```
        return 1;
}



void create(struct tree_node *root)    // void συναρτηση που εκτελείται για κάθε κόμβο
{
        int status;
        pid_t p;
        int i;
       // το for εκτελείται τόσες φορές όσα είναι τα παιδιά του κόμβου που δείχνει ο  root
        for (i=0; i<root->nr_children; i++) {

                fprintf(stderr, "Parent, PID = %ld: Creating child...\n",
                (long)getpid());

                p = fork();             // δημιούργησε το παιδί
                if (p < 0) {
                        /* fork failed */
                        perror("fork");
                        exit(1);
                }
                if (p == 0) {
                        /* In child process */
                        change_pname((root->children+i)->name);   // ονόμασέ το με το
όνομα του αντίστοιχου κόμβου
                       // αν το παιδί έχει παιδιά τότε κάλεσε ξανά τη συνάρτηση με δείκτη που
δείχνει στο παιδί (που φτιάχτηκε πριν λίγο)
                        if ((root->children+i)->nr_children>0)
                                create(root->children+i);
                        // αν δεν έχει παιδιά είναι φύλλο, οπότε θα κοιμηθεί
                        else
                                sleep(5);
                        exit(101);          // τερματίζει
                        }

                printf("Parent, PID = %ld: Created child with PID = %ld, waiting
                for it to terminate...\n", (long)getpid(), (long)p);
                }
          // περιμένει να τερματιστούν όλα τα παιδιά του πριν τερματίσει
```

```
                for (i=0; i<root->nr_children; i++) {
                        p=wait(&status);
                        explain_wait_status(p, status);
                }
                printf("Parent %s: All done, exiting...\n", root->name);

                return;
}
```

Παράδειγμα Εκτέλεσης:

```
oslabe15@anafi:~/erg2/forktree$ ./ask2
Usage: ./ask2 <input_tree_file>

oslabe15@anafi:~/erg2/forktree$ ./ask2 proc.tree
A
      B
            E
            F
      C
      D

Parent Father, PID = 3595: Creating child...
Parent, PID = 3596: Creating child...
Parent, PID = 3596: Created child with PID = 3597, waiting for it to terminate...
Parent, PID = 3596: Creating child...
Parent, PID = 3596: Created child with PID = 3598, waiting for it to terminate...
Parent, PID = 3596: Creating child...
Parent, PID = 3596: Created child with PID = 3599, waiting for it to terminate...
Parent, PID = 3597: Creating child...
Parent, PID = 3597: Created child with PID = 3600, waiting for it to terminate...
Parent, PID = 3597: Creating child...
Parent, PID = 3597: Created child with PID = 3601, waiting for it to terminate...


A(3596)─┬─┬─B(3597)─┬─┬─E(3600)
        │ │         └──F(3601)
        │ ├──C(3598)
        │ └──D(3599)


My PID = 3596: Child PID = 3598 terminated normally, exit status = 101
My PID = 3596: Child PID = 3599 terminated normally, exit status = 101
My PID = 3597: Child PID = 3600 terminated normally, exit status = 101
My PID = 3597: Child PID = 3601 terminated normally, exit status = 101
```

```
Parent B: All done, exiting...
My PID = 3596: Child PID = 3597 terminated normally, exit status = 101
Parent A: All done, exiting...
My PID = 3595: Child PID = 3596 terminated normally, exit status = 101
```

Απαντήσεις στις ερωτήσεις:

1. Με ποια σειρά εμφανίζονται τα μηνύματα έναρξης και τερματισμού των διεργασιών; γιατί;

Στο πρόγραμμα τυπώνουμε ενημερωτικά μηνύματα όταν δημιουργείται ένα παιδί-διεργασία ή όταν πεθαίνει κάποιο. Προφανώς η αρχική διεργασία-πατέρας επειδή είναι εκείνη που δημιουργεί τις άλλες θα είναι και αυτή που τυπώνει μήνυμα πρώτη, όταν δημιουργεί την πρώτη διεργασία, και θα ναι και αυτή που τυπώνει τελευταία μήνυμα, όταν πεθαίνει. Τώρα σε ό,τι αφορά τις ενδιάμεσες διεργασίες, εμφανίζουν μήνυμα με τη σειρά που δημιουργούνται, αλλά δεν μπορούμε να ξέρουμε ακριβώς τη σειρά. Εξαρτάται από το πόσους υπολογισμούς εκτελεί η καθεμία. Βέβαια αυτό ισχύει για τις διεργασίες που στο δέντρο διεργασιών είναι στο ίδιο “βάθος”, γιατί σε διαφορετικό βάθος είναι προφανές ότι αυτές που είναι πιο βαθιά θα δημιουργηθούν πιο αργά, γιατί πρέπει πρώτα να δημιουργηθούν οι γονείς τους, και θα κλείσουν και πρώτες, γιατί πρέπει οι γονείς τους να κάνουν wait() .

Άσκηση 1.3

Source Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <assert.h>
#include <sys/wait.h>

#include "proc-common.h"
#include "tree.h"


void create(struct tree_node *root);
```

```c
int main(int argc, char *argv[])
{
        struct tree_node *root;
        // αν τα αρχεία εισόδου είναι διάφορα του 2 τότε τύπωσε το ακόλουθο μήνυμα
        if (argc != 2) {
                fprintf(stderr, "Usage: %s <input_tree_file>\n\n", argv[0]);
                exit(1);
        }

        root = get_tree_from_file(argv[1]);       // ο δείκτης root δείχνει στη ρίζα του δέντρου που διαβάζουμε από το αρχείο εισόδου

        if (root!=NULL) {

                print_tree(root);                 // τύπωσε τα στοιχεία του δέντρου
                printf("\n");

                pid_t pid;
                int status;

                fprintf(stderr, "Parent Father, PID = %ld: Creating
                child...\n",(long)getpid());

                pid = fork();          // δημιούργησε το παιδί που αντιστοιχεί στη ρίζα του δέντρου
                if (pid < 0) {
                        perror("main: fork");
                        exit(1);
                }
                if (pid == 0) {
                        /* Child */
                        change_pname(root->name);       // ονόμασέ το με το όνομα του κόμβου

                        create(root);
                        exit(101);
                }

                wait_for_ready_children(1);       // περιμένει να λάβει σήμα από το παιδί της
```

```
                show_pstree(pid);                       // εμφάνισε το δέντρο

                kill(pid, SIGCONT);                     // στείλε σήμα στο παιδί να συνεχίσει

                wait(&status);                          // περιμένει να τερματιστεί το παιδί της
                explain_wait_status(pid, status);
        }
        else                // αν το δέντρο είναι κενό τύπωσε το αντίστοιχο μήνυμα
                printf("The input tree file is empty!\n");

        return 1;
}


void create(struct tree_node *root)    // void συναρτηση που εκτελείται για κάθε κόμβο
{
        int status;
        pid_t A[200];
        int i;

       // το for εκτελείται τόσες φορές όσα είναι τα παιδιά του κόμβου που δείχνει ο  root
        for (i=0; i<root->nr_children; i++) {

                fprintf(stderr, "Parent, PID = %ld: Creating
                child...\n",(long)getpid());

                A[i] = fork();              // δημιούργησε το παιδί
                if (A[i] < 0) {
                        /* fork failed */
                        perror("fork");
                        exit(1);
                }
                if (A[i] == 0) {
                        /* In child process */
                        change_pname((root->children+i)->name);    // ονόμασέ το με το
όνομα του αντίστοιχου κόμβου

                        create(root->children+i);   // η συνάρτηση είναι αναδρομική
                        exit(101);
                }
                printf("Parent, PID = %ld: Created child with PID = %ld, waiting
```

```
                for it to terminate...\n", (long)getpid(), (long)A[i]);
        }

        wait_for_ready_children(root->nr_children);     // περιμένει να λάβει σήμα από όλα
τα παιδιά του ότι ανέστειλλαν τη λειτουργία τους
        raise(SIGSTOP);          // η διεργασία στέλνει σήμα στον εαυτό της να σταματήσει
μέχρι να λάβει σήμα για να συνεχίσει
        printf("PID = %ld, name = %s is awake\n",(long)getpid(), root->name);
        // το for εκτελείται για όλα τα παιδιά, στέλνει σήμα στο ένα παιδί να συνεχίσει και περιμένει τον
τερματισμό του (επιτυγχάνεται έτσι η κατά βάθος εκτύπωση μηνυμάτων)
        for (i=0; i<root->nr_children; i++) {
                kill(A[i],SIGCONT);
                wait(&status);
                explain_wait_status(A[i], status);
        }
        return;
}
```

Παράδειγμα Εκτέλεσης:

```
oslabe15@anafi:~/erg2/forktree$ ./ask3 proc.tree
A
      B
            E
            F
      C
      D

Parent Father, PID = 3609: Creating child...
Parent, PID = 3610: Creating child...
Parent, PID = 3610: Created child with PID = 3611, waiting for it to
terminate...
Parent, PID = 3610: Creating child...
Parent, PID = 3610: Created child with PID = 3612, waiting for it to
terminate...
Parent, PID = 3610: Creating child...
Parent, PID = 3610: Created child with PID = 3613, waiting for it to
terminate...
My PID = 3610: Child PID = 3613 has been stopped by a signal, signo = 19
My PID = 3610: Child PID = 3612 has been stopped by a signal, signo = 19
Parent, PID = 3611: Creating child...
Parent, PID = 3611: Created child with PID = 3614, waiting for it to
terminate...
```

```
Parent, PID = 3611: Creating child...
Parent, PID = 3611: Created child with PID = 3615, waiting for it to
terminate...
My PID = 3611: Child PID = 3615 has been stopped by a signal, signo = 19
My PID = 3611: Child PID = 3614 has been stopped by a signal, signo = 19
My PID = 3610: Child PID = 3611 has been stopped by a signal, signo = 19
My PID = 3609: Child PID = 3610 has been stopped by a signal, signo = 19


A(3610)─┬─B(3611)─┬─E(3614)
        │         └─F(3615)
        ├─C(3612)
        └─D(3613)


PID = 3610, name = A is awake
PID = 3611, name = B is awake
PID = 3614, name = E is awake
My PID = 3611: Child PID = 3614 terminated normally, exit status = 101
PID = 3615, name = F is awake
My PID = 3611: Child PID = 3615 terminated normally, exit status = 101
My PID = 3610: Child PID = 3611 terminated normally, exit status = 101
PID = 3612, name = C is awake
My PID = 3610: Child PID = 3612 terminated normally, exit status = 101
PID = 3613, name = D is awake
My PID = 3610: Child PID = 3613 terminated normally, exit status = 101
My PID = 3609: Child PID = 3610 terminated normally, exit status = 101
```

Απαντήσεις στις ερωτήσεις:

1. Στις προηγούμενες ασκήσεις χρησιμοποιήσαμε τη sleep() για τον συγχρονισμό των διεργασιών. Τι πλεονεκτήματα έχει η χρήση σημάτων;

Τα σήματα είναι ένας τρόπος διαδεργασιακής επικοινωνίας (ipc). Εμείς στο πρόγραμμα που γράψαμε στην 1.2 θέλαμε η show_pstree() να εμφανίσει το δέντρο όταν όλα τα παιδιά είναι ανοιχτά, για να τα δούμε, πριν κάποιο τερματίσει και δεν φανεί, οπότε βάλαμε στα παιδιά-φύλλα και στη ρίζα να εκτελέσουν τη συνάρτηση sleep(), για να εξασφαλίσουμε ότι δεν θα κλείσουν πρόωρα. Ωστόσο αυτή η μέθοδος είναι πολύ αναξιόπιστη γιατί πολύ απλά δεν ξέρουμε πάντα πόσο χρόνο θα κάνει μια διεργασία που εκτέλει έναν υπολογισμό να τερματίσει, και επίσης δεν θέλουμε να παγώνουμε διεργασίες χωρίς λόγο. Με τη χρήση σημάτων όμως είναι πολύ εύκολο να ενημερώσουμε μια άλλη διεργασία, στέλνοντας απλά σήμα. Η διαφορά αυτή

φάνηκε άμεσα στην παρούσα άσκηση, καθώς όπως είδαμε η εκτέλεση ήταν ακαριαία. Έτσι επιτύχαμε πραγματικό συγχρονισμό ανάμεσα στις διεργασίες.

2. Ποιος ο ρόλος της wait_for_ready_children(); Τι εξασφαλίζει η χρήση της και τι πρόβλημα θα δημιουργούσε η παράλειψή της;

Η δοθείσα συνάρτηση wait_for_ready_children() παγώνει μία διεργασία μέχρι τα παιδιά της να κάνουν raise(SIGSTOP). Στο πρόβλημα μας εξασφαλίζει ότι οι γονείς θα συνεχίσουν την εκτέλεση τους μόνο όταν όλα τα παιδιά τους έχουν διακόψει προσωρινά τη λειτουργία τους. Έτσι, η διεργασία που την καλεί μπορεί στη συνέχεια να διακόψει προσωρινά και αυτή τη λειτουργία της. Η παράλειψη χρησιμοποίησης της θα οδηγήσει σε απώλεια του συγχρονισμού των διεργασιών.

Άσκηση 1.4

Source Code:

```c
#include <stdio.h>
#include <stdlib.h>
#include <unistd.h>
#include <signal.h>
#include <assert.h>
#include <sys/wait.h>

#include "proc-common.h"
#include "tree.h"


void create(struct tree_node *root, int fd);


int main(int argc, char *argv[])
{
        struct tree_node *root;
        // αν τα αρχεία εισόδου είναι διάφορα του 2 τότε τύπωσε το ακόλουθο μήνυμα
        if (argc != 2) {
                fprintf(stderr, "Usage: %s <input_tree_file>\n\n", argv[0]);
                exit(1);
        }

        root = get_tree_from_file(argv[1]);        // ο δείκτης root δείχνει στη ρίζα του δέντρου που διαβάζουμε από το αρχείο εισόδου
```

```c
        if (root!=NULL) {

                print_tree(root);       // τύπωσε τα στοιχεία του δέντρου
                printf("\n");

                pid_t pid;
                int status;
                int pfd[2];

                printf("Parent Father: Creating pipe...\n");
                if (pipe(pfd) < 0) {            // φτιάξε σωλήνωση
                        perror("pipe");
                        exit(1);
                }

                fprintf(stderr, "Parent Father, PID = %ld: Creating
                child...\n",(long)getpid());

                pid = fork();  // δημιούργησε το παιδί που αντιστοιχεί στη ρίζα του δέντρου
                if (pid < 0) {
                        perror("main: fork");
                        exit(1);
                }
                if (pid == 0) {
                        /* Child */
                        close (pfd[0]);// κλείσε το άκρο ανάγνωσης της σωλήνωσης για το
παιδί
                        change_pname(root->name);// ονόμασέ το με το όνομα του κόμβου
                        create(root, pfd[1]);
                        close (pfd[1]);     // κλείσε και το άκρο εγγραφής για το παιδί
                        exit(100);
                }

                close (pfd[1]);  // κλείσε το άκρο εγγραφής για τη διεργασία-πατέρα
                sleep(3);// η αρχική διεργασία κοιμάται περιμένοντας να δημιουργηθούν όλες οι
διεργασίες


                show_pstree(pid);       // εμφάνισε το δέντρο

                wait(&status);          // περιμένει να τερματιστεί το παιδί της
                explain_wait_status(pid, status);
```

```
                int val;
                // διάβασε την τελική τιμή της πράξης από τη σωλήνωση
                if (read(pfd[0], &val, sizeof(val)) != sizeof(val)) {
                        perror("Father: read from pipe");
                        exit(1);
                }
                printf("Father: received value from the pipe! The result is:
                %d\n", val);           // τύπωσε την τελική τιμή της πράξης
                close (pfd[0]);        // κλείσε το άκρο ανάγνωσης για τη διεργασία-πατέρα
        }
        else                // αν το δέντρο είναι κενό τύπωσε το αντίστοιχο μήνυμα
                printf("The input tree file is empty!\n");

        return 1;
}



void create(struct tree_node *root, int fd) // void συναρτηση που εκτελείται για κάθε κόμβο
{
        int status;
        pid_t p;
        int i, val1, val2;
        int pfd1[2], pfd2[2];
        // φτιάξε δύο σωληνώσεις, μία για κάθε παιδί
        if (root->nr_children>0) {

                printf("Parent: Creating pipe...\n");
                if (pipe(pfd1) < 0) {
                        perror("pipe");
                        exit(1);
                }

                printf("Parent: Creating pipe...\n");
                if (pipe(pfd2) < 0) {
                        perror("pipe");
                        exit(1);
                }
        }
```

```
        // το for εκτελείται τόσες φορές όσα είναι τα παιδιά του κόμβου που δείχνει ο  root
        for (i=0; i<root->nr_children; i++) {

                fprintf(stderr, "Parent, PID = %ld: Creating child...\n",
                (long)getpid());

                p = fork();             // δημιούργησε μία διεργασία-παιδί
                if (p < 0) {
                        /* fork failed */
                        perror("fork");
                        exit(1);
                }
                if (p == 0) {
                        /* In child process */
                        change_pname((root->children+i)->name);    // ονόμασέ το με το
όνομα του κόμβου
                       // κλείσε τα άκρα ανάγνωσης των σωληνώσεων για καθέ ένα παιδί
                        close(pfd1[0]);
                        close(pfd2[0]);
                        // αν έχει παιδιά ο κόμβος κάλεσε την ίδια συνάρτηση
                        if ((root->children+i)->nr_children>0) {
                                if (0==i)
                                        create(root->children+i, pfd1[1]);
                                else
                                        create(root->children+i, pfd2[1]);
                        }
                        // αλλιώς γράψε στη σωλήνωση
                        else {
                                if (0==i) {     // αν είναι το πρώτο παιδί
                                        int k=atoi((root->children+i)->name);
                                        if (write(pfd1[1], &k, sizeof(k)) !=
sizeof(k)) {
                                                perror("child: write to pipe");
                                                exit(1);
                                        }
                                }
                                else {      // αν είναι το δεύτερο παιδί
                                        int k=atoi((root->children+i)->name);
                                        if (write(pfd2[1], &k, sizeof(k)) !=
sizeof(k)) {
                                                perror("child: write to pipe");
```

```
                                        exit(1);
                                }
                        }       // αφού είναι φύλλο θα κοιμηθεί
                        sleep(5);
                }
                // κλείσε τα άκρα εγγραφής για το παιδί
                close (pfd1[1]);
                close (pfd2[1]);
                exit(101);
        }
        printf("Parent, PID = %ld: Created child with PID = %ld, waiting
        for it to terminate...\n", (long)getpid(), (long)p);
}
close(pfd1[1]);     // κλείσε τα άκρα εγγραφής για τη διεργασία-πατέρα
close(pfd2[1]);
// περίμενε να τερματιστούν όλα τα παιδιά
for (i=0; i<root->nr_children; i++) {
        p=wait(&status);
        explain_wait_status(p, status);
}

// διάβασε τους αριθμούς από τη σωλήνωση
printf("Parent: My PID is %ld. Receiving an integer value.\n",
(long)getpid());
if (read(pfd1[0], &val1, sizeof(val1)) != sizeof(val1)) {
        perror("parent: read from pipe");
        exit(1);
}
printf("Parent %ld: received value %d from the pipe.\n",(long)getpid(),
val1);

printf("Parent: My PID is %ld. Receiving an integer value.\n",
(long)getpid());
if (read(pfd2[0], &val2, sizeof(val2)) != sizeof(val2)) {
        perror("parent: read from pipe");
        exit(1);
}
printf("Parent %s: received value %d from the pipe. Will now compute.\n",
root->name, val2);

// κάνε την πράξη που αναφέρεται στον κόμβο
```

```
        int res;
        if (root->name[0]=='+')
                res=val1+val2;
        else
                res=val1*val2;

        // γράψε το αποτέλεσμα στη σωλήνωση με τον πατέρα της διεργασίας που εκτελείται
        if (write(fd, &res, sizeof(res)) != sizeof(res)) {
                        perror("parent: write to pipe");
                        exit(1);
        }
         // κλείσε τα άκρα ανάγνωσης της σωλήνωσης της διεργασίας-πατέρα (που εκτελείται τώρα)
        close (pfd1[0]);
        close (pfd2[0]);

        printf("Parent %s: All done, exiting...\n", root->name);
        return;
}
```

Παράδειγμα Εκτέλεσης:

```
oslabe15@anafi:~/erg2/forktree$ ./ask41 expr.tree
+
      10
      *
            +
                  5
                  7
            4

Parent Father: Creating pipe...
Parent Father, PID = 3639: Creating child...
Parent: Creating pipe...
Parent: Creating pipe...
Parent, PID = 3640: Creating child...
Parent, PID = 3640: Created child with PID = 3641, waiting for it to
terminate...
Parent, PID = 3640: Creating child...
Parent, PID = 3640: Created child with PID = 3642, waiting for it to
terminate...
Parent: Creating pipe...
```

```
Parent: Creating pipe...
Parent, PID = 3642: Creating child...
Parent, PID = 3642: Created child with PID = 3643, waiting for it to
terminate...
Parent, PID = 3642: Creating child...
Parent, PID = 3642: Created child with PID = 3644, waiting for it to
terminate...
Parent: Creating pipe...
Parent: Creating pipe...
Parent, PID = 3643: Creating child...
Parent, PID = 3643: Created child with PID = 3645, waiting for it to
terminate...
Parent, PID = 3643: Creating child...
Parent, PID = 3643: Created child with PID = 3646, waiting for it to
terminate...


+(3640)─┬──*(3642)─┬──+(3643)─┬─5(3645)
        │          │          └─7(3646)
        │          └─4(3644)
        └─10(3641)


My PID = 3640: Child PID = 3641 terminated normally, exit status = 101
My PID = 3642: Child PID = 3644 terminated normally, exit status = 101
My PID = 3643: Child PID = 3645 terminated normally, exit status = 101
My PID = 3643: Child PID = 3646 terminated normally, exit status = 101
Parent: My PID is 3643. Receiving an integer value.
Parent 3643: received value 5 from the pipe.
Parent: My PID is 3643. Receiving an integer value.
Parent +: received value 7 from the pipe. Will now compute.
Parent +: All done, exiting...
My PID = 3642: Child PID = 3643 terminated normally, exit status = 101
Parent: My PID is 3642. Receiving an integer value.
Parent 3642: received value 12 from the pipe.
Parent: My PID is 3642. Receiving an integer value.
Parent *: received value 4 from the pipe. Will now compute.
Parent *: All done, exiting...
My PID = 3640: Child PID = 3642 terminated normally, exit status = 101
Parent: My PID is 3640. Receiving an integer value.
Parent 3640: received value 10 from the pipe.
Parent: My PID is 3640. Receiving an integer value.
Parent +: received value 48 from the pipe. Will now compute.
Parent +: All done, exiting...
My PID = 3639: Child PID = 3640 terminated normally, exit status = 100
Father: received value from the pipe! The result is: 58
```

Απαντήσεις στις ερωτήσεις:

1. Πόσες σωληνώσεις χρειάζονται στη συγκεκριμένη άσκηση ανά διεργασία; Θα μπορούσε κάθε γονική διεργασία να χρησιμοποιεί μόνο μία σωλήνωση για όλες τις διεργασίες παιδιά; Γενικά, μπορεί για κάθε αριθμητικό τελεστή να χρησιμοποιηθεί μόνο μια σωλήνωση;

Στη συγκεκριμένη άσκηση, έχουμε χρησιμοποιήσει δύο σωληνώσεις ανά διεργασία. Θα μπορούσε κάθε γονική διεργασία να χρησιμοποιεί μόνο μία σωλήνωση, καθώς οι τελεστές “+” και “*” υποστηρίζουν την αντιμεταθετική ιδιότητα, οπότε δεν επηρεάζει η σειρά που θα διαβάσει η γονική διεργασία τους αριθμούς από την σωλήνωση. Όμως, δεν μπορεί να χρησιμοποιηθεί μόνο μία σωλήνωση για κάθε αριθμητικό τελεστή, όπως στην περίπτωση της διαίρεσης ή της αφαίρεσης όπου η σειρά που θα διαβαστούν οι αριθμοί έχει σημασία. Σε τέτοια περίπτωση θα μπορούσαμε να χρησιμοποιήσουμε κάποια μορφή συγχρονισμού (π.χ. σήματα) αν θέλαμε να το υλοποιήσουμε με μία σωλήνωση.

2. Σε ένα σύστημα πολλαπλών επεξεργαστών, μπορούν να εκτελούνται παραπάνω από μια διεργασίες παράλληλα. Σε ένα τέτοιο σύστημα, τι πλεονέκτημα μπορεί να έχει η αποτίμηση της έκφρασης από δέντρο διεργασιών, έναντι της αποτίμησης από μία μόνο διεργασία;

Έχει το πλεονέκτημα της ταχύτητας η οποία επιτυγχάνεται ακριβώς χάρη στη παράλληλη εκτέλεση.